GCCS/DII COE System Integration Support

Technical Report/Study: GCCS Strategic Technical Architecture (Final)

February 27, 1997

Prepared for:

DISA/JEJA
ATTN: Ms. Claire Burchell
45335 Vintage Park Plaza
Sterling, VA 20166-6701

Contract Number: DCA 100-94-D-0014
Delivery Order Number: 330, Task 2
CDRL Number: A005

Prepared by:

Computer Sciences Corporation
Defense Enterprise Integration Services
Four Skyline Place
5113 Leesburg Pike, Suite 700
Falls Church, VA 22041

THIS DOCUMENT IS UNCLASSIFIED

TABLE OF CONTENTS

List of Figures

Preface

The following conventions are used in this document:

Bold	Used for information that is typed, pressed, or selected in executables and instructions. For example, select**connect to host**.
Italics	Used for file names, directories, scripts, commands, user IDs, document names, and Bibliography references; and any unusual computerese the first time it is used in text.
Underline	Used for emphasis.
Arrows <>	Used to identify keys on the keyboard. For example <Return>.
AQuotation Marks@	Used to identify informal, computer-generated queries and reports, or coined names; and to clarify a term when it appears for the first time. For example ΑData-Generation Report@
`Courier Font`	Used to denote anything as it appears on the screen or command lines. For example `tar xvf dev/rmt/3mm`.
Capitalization	Used to identify keys, screen icons, screen buttons, field, and menu names.

1.0 INTRODUCTION

The purpose of this study is to predict the technological makeup of the Global Command and Control System (GCCS) 12 to 18 months in the future. When this study began, GCCS had already incorporated some concepts from the Internet into the system. GCCS Web pages are available on the Secret Internet Protocol Router Network (SIPRNET) and the Netscape browser is a standard component of a GCCS installation. The ability to locate information over the Internet including database searches through extensive use of HyperText Transfer Protocol (HTTP)/HyperText Markup Language (HTML) Web pages could be conceptually extended to GCCS as a way to fulfill mission area requirements.

In May 1995, Sun Microsystems introduced the Java programming as a way to create dynamic Web pages. Since that time, Java has caught the information technology industry's attention. Developments in the Java area are occurring at such speed that reference material a few months old is already out of date. A search of the Internet yields so many references to Java that the future picture is somewhat confusing. Because of the great potential impact of Java on network-centric design, the primary focus of the study became using Java in GCCS. This report summarizes the preliminary findings of network-centric technology applied to GCCS, especially the use of Java.

GCCS, in its current implementation, is a network-oriented system. For sites that do not have a database server local to their computing domain, the network is an essential resource. Even sites that have a database server depend upon a Local Area Network (LAN) to allow their users to access the database and depend upon the Wide Area Network (WAN) (i.e., SIPRNET) to access backup database servers in case their own goes down. Given that GCCS is a distributed network-oriented system made up of many applications, it becomes both desirable (in terms of making use of an available resource) and essential (this resource is limited) to make intelligent use of the network (making the bandwidth perform useful work) and the distributed system (load balancing, fault detection, and error recovery).

GCCS is a large scale long-lived system comprised of multiple independently purchased commercial off-the-shelf (COTS) and multiple independently developed application programs whose clients must run on all platforms. The System must continue to function across hardware repurchases. This environment places restrictions on software developed for GCCS. Software developed for GCCS must use:

- Standard language and libraries to ease porting across platforms.

- Open standards that are supported by the majority of vendors.

- Well-defined application interfaces allowing independent use for developing and upgrading applications.

- Popular standards that cover the products of many vendors with a large installed base in the commercial sector, ensuring that commercial support for the standard does not go away.

The World Wide Web (WWW) has revolutionized computing. The communication network is now an organic asset, which is as important to a site or a user's computation as the machine on their desk. GCCS has taken advantage of the Web both to distribute data and to implement some simple applications. In the last two years, Java and a new network programming construct (the "Applet") have become popular. This popularity stems from the fact that Java and Java Applets allow executable programs to be downloaded over the network via HTTP and then executed on the local machine in a mode that is transparent to the user.

The rapid increase in the use of Java has been accompanied by rapid changes and growth in Java itself and in the Java/Applet development support marketplace. There is an ever-growing array of COTS products designed to support Java/Applet development and the integration of Java code with servers and databases. Java Virtual Machines produced by different vendors are maturing, and Web browsers are being equipped with AJust-In-Time@ (JIT) compilers, which allow Applets to execute close to the execution speed of native applications. Finally, the language itself is acquiring new features as the growing user community demands the features that will allow Java to support their applications.

The use of Java (in conjunction with the Web) in GCCS has the potential to solve some of the most significant problems GCCS is facing:

C The move to object-oriented design and analysis to improve code re-use and maintainability.

C The need to support a wide array of computing platforms. Defense Information Infrastructure (DII) cannot Alock out@ any single vendor, however an attempt by Defense Information Systems Agency (DISA) to support binary executables on all platforms will result in unacceptable and unsustainable growth in developer effort.

C Software testing for all applications on all platforms is not feasible.

C Software distribution and configuration management for all applications on all platforms becomes impossible. Applets, on the other hand, can be stored and updated in a single location and dynamically downloaded when the user executes them.

C A common user interface has not yet been effectively implemented in GCCS. In a network-centric GCCS users will use their favorite web browser to access Applets, and these Applets will have a common look and feel.

Based on our study, we predict that within 18 to 24 months, Java and Java Applets should be playing a significant role in the implementation of the GCCS Common Operating Environment (COE) and its mission applications. We use the word Ashould@ since widespread use of Java and network-centric technologies will impact the current approach to DII, and the changes required will be difficult to accept. The introduction of the network-centric concepts (including Java) need to be phased into GCCS and must co-exist with DII. If the stirring in the commercial world about Java and planned development using Java are any indicator of the future, introduction of Java into GCCS will be hard to resist.

2.0 THE ROLE OF JAVA IN A NETWORK-CENTRIC ARCHITECTURE

With the introduction of the WWW, it became possible to see graphical pages sent across the Internet by Web servers. These Web pictures can include sound, video, and text as well as hypertext links to related pages. A Web browser, such as Netscape, provides an easy-to-use interface for traveling around the Web and visiting other people=s pages. An HTML editor and a Web server are the essential components for building Web pages that can be accessed by network users. While the HTML-based Web pages are an improvement from using complex UNIX-based commands (i.e., ftp, news, gopher, wais, and telnet) to get around the net, they have the drawback that they are static. The alternative to static pages are interactive pages that dynamically change themselves according to feedback from the user. These pages are programmed to accept input from the user, compute results, and then display them.

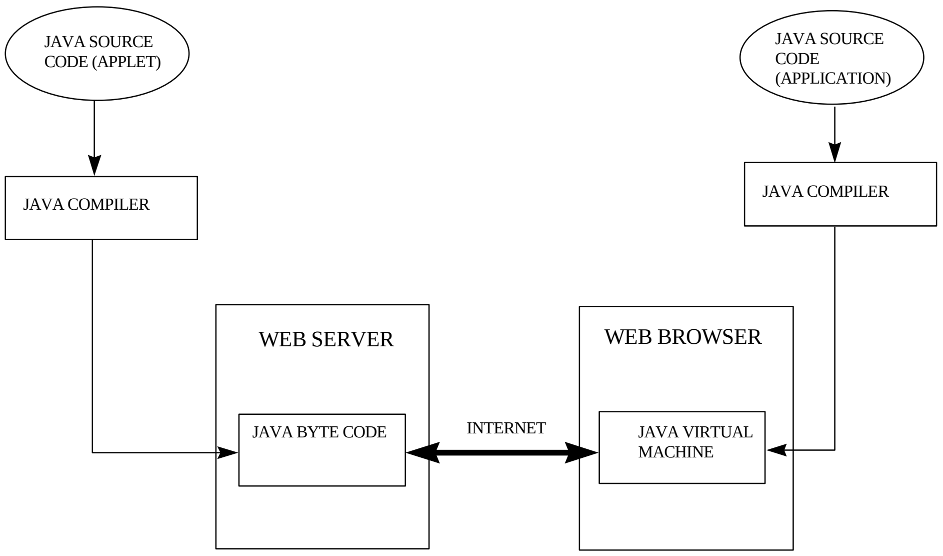
Dynamic Web pages can be created using Java, which is the essential role of Java on the Web. Java is a full-featured programming language that allows programmers to compose executable content for the Web. The Java language is designed to be usable on all platforms. This is an essential feature for a programming language with executable content since executable content is only useful if it can be executed on all platforms without porting and recompiling.

Java can be used in two modes: The first is the use of Java as down-loadable executable content. In this mode, Java programs (which are called Applets) reside on a Web server and are loaded across the network to execute on a user=s machine. The second is to use Java as any other compilable language that is resident on a user machine. These programs are called Java Applications. In the case of Java Applets and Java Applications, both will execute on a Java Virtual Machine (VM).

A Java VM interprets Java instructions and translates them into machine-specific instructions. Java instructions are fixed length consisting of opcodes and arguments and are very similar to assembly language instructions. The difference being that these instructions execute on the Java VM so that they will execute on any machine running the Java VM. Since the Java instructions are compiled into 8-bit opcodes and arguments, they are calledAbytecode.@ This approach allows Java to be run on many different kinds of machines. Use of the Java VM allows Java to get around problems that have plagued the C programming language, which is not nearly as platform independent as it was intended to be. Java programs are not hampered by machine-dependent structures such as byte ordering (low or high endian), pointer size (16 or 32 bit), and integer size (8 bit, 16 bit, or 32 bit).

Each Java VM is written to a specific computing platform and translates the more generic Java instructions into platform-specific instructions. Netscape Navigator 2.0.x and later, and Microsoft Internet Explorer 3.0 each include a complete Java VM that can interpret the Java bytecode. The browser is also responsible for enforcing Java security rules, and each browser has its own security manager.

Figure 2-1 shows the essential elements in using Java Applets and Applications. Applet programs are compiled into Java bytecode, which resides on a Web server. When a user browses a Web page containing Applets, they are automatically downloaded to the Web browser where they are executed on the Java VM. Java applications are compiled into bytecode, which is resident on the local machine and is executed on the Java virtual machine of the Web browser. Java as an application language is an object-oriented language and is not governed by the same security restrictions as Java Applets.

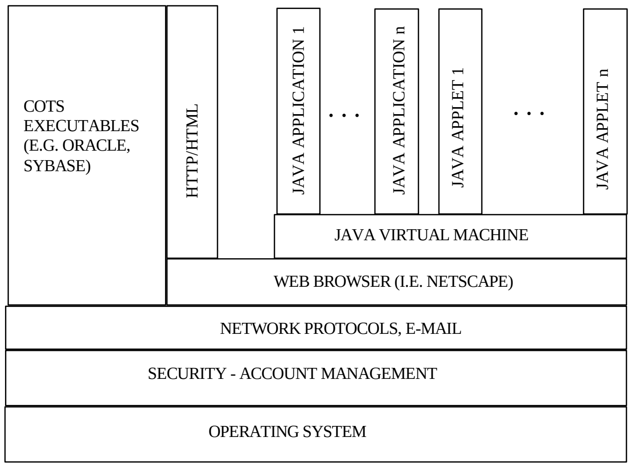

JAVA SOURCE CODE (APPLET)
JAVA COMPILER
JAVA SOURCE CODE (APPLICATION)
JAVA COMPILER
WEB SERVER
JAVA BYTE CODE
INTERNET
WEB BROWSER
JAVA VIRTUAL MACHINE

3.0 NETWORK-CENTRIC ARCHITECTURE FOR GCCS

A network-centric architecture for GCCS would emulate the operation of the Internet using the SIPRNET. The popularity (and utility) of the Internet has increased substantially since the introduction of Web technology and graphical interfaces. In early 1990, few people used the Internet, by the beginning of 1995 there were slightly more than 200,000 users and by July 1996 there were 12,000,000 users. This rapid growth can only be explained by the ease of use made possible by the Web and modern browsers. This widespread and growing use of the Internet drives the commercial marketplace to develop low-cost software technology. The introduction of Java means that much of this software will be platform independent.

3.1 Components of Network-Centric Architecture

Figure 3-1 shows the significant components of a network-centric architecture. The components shown in Figure 3-1 illustrate the Anetwork only@ components and show a significant involvement of Java. In the figure all applications are either Java applications or Java Applets. HTTP/HTML also figures prominently for building Web pages.

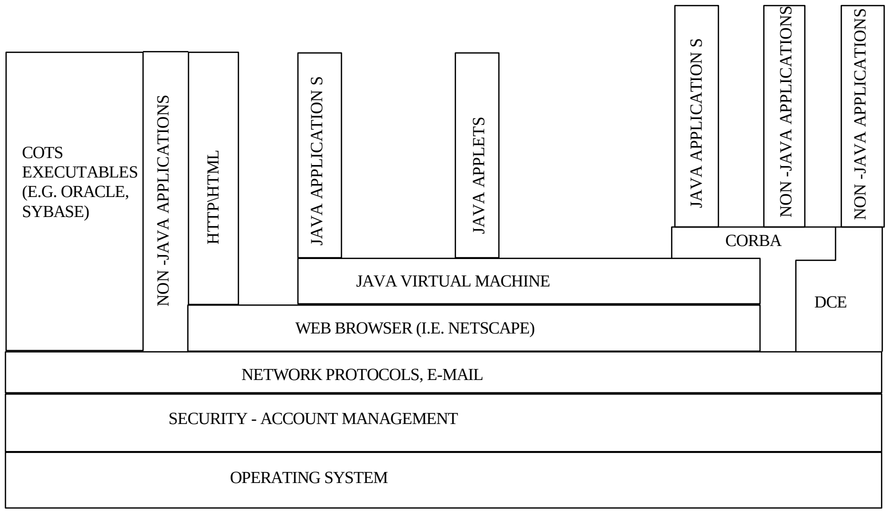

COTS EXECUTABLES (E.G. ORACLE, SYBASE)
HTTP/HTML
JAVA APPLICATION 1
. . .
JAVA APPLICATION n
JAVA APPLET 1
. . .
JAVA APPLET n
JAVA VIRTUAL MACHINE
WEB BROWSER (I.E. NETSCAPE)
NETWORK PROTOCOLS, E-MAIL
SECURITY - ACCOUNT MANAGEMENT
OPERATING SYSTEM

In the all-network approach shown in Figure 3-1, the low-level COE is made up of the Java VM and the browser. Operating system dependence is reduced since the browser is designed to work with the operating system to provide essential computing services such as printing. The importance of COE standards (such as X-windows and Motif) is overshadowed by the Java VM, which provides the necessary graphics objects for designing the user interface. The use of the Java VM ensures that application or Applet code written in Java will execute on the platform. The browser can serve as a launch window for the applications (and Applets) and provides basic desktop functions.

Figure 3-1 shows a simplified architecture and does not take into account common applications such as JMTK or the common tactical picture. These would be supplied as part of a COE however, they could be written in Java. It is important to remember that Java is a standalone object-oriented language and that current applications written in other languages could be re-written in Java. Java applications executing on Web servers and/or client workstations as well as Applets transferred via the net provide some interesting implementation possibilities.

Within the next 18 months however, it is unlikely that Java applications will replace current GCCS functionality since the investment in legacy code and in standard code development will dictate that these applications be maintained. The DII COE will also proliferate during the next 18 months so that GCCS architectures will continue to use the DII COE however, the use of network-centric technology and Java will increase and will co-exist with existing software implementations.

Figure 3-2 illustrates the major architectural components found in a hybrid architecture including current DII, legacy, and network-centric applications; Common Object Request Broker Architecture (CORBA); and Distributed Computer Environment (DCE). Since Java is a full-featured standalone object-oriented language it can use CORBA in the same manner as C/C++ and there are commercial developments underway to do just that.



3.2 Topology of GCCS in a Network-Centric Architecture

The original architecture proposed for GCCS was a three-tier architecture. However, GCCS is currently a two-tier architecture consisting of clients with resident applications (which combine the application tier and the user interface tier) and database servers. The clients in GCCS are primarilyAfat clients@ with large disk storage capacity and many resident applications. While smaller client workstations exist, they tend to have applications installed locally. Even the use of NFS mounting in GCCS is limited primarily to those sites with experienced system administrators.

The network-centric architecture will be a three-tier architecture. It will have fat and thin clients, Web servers, and database servers. Figure 3-3 illustrates the topology needed by GCCS to use network technology.

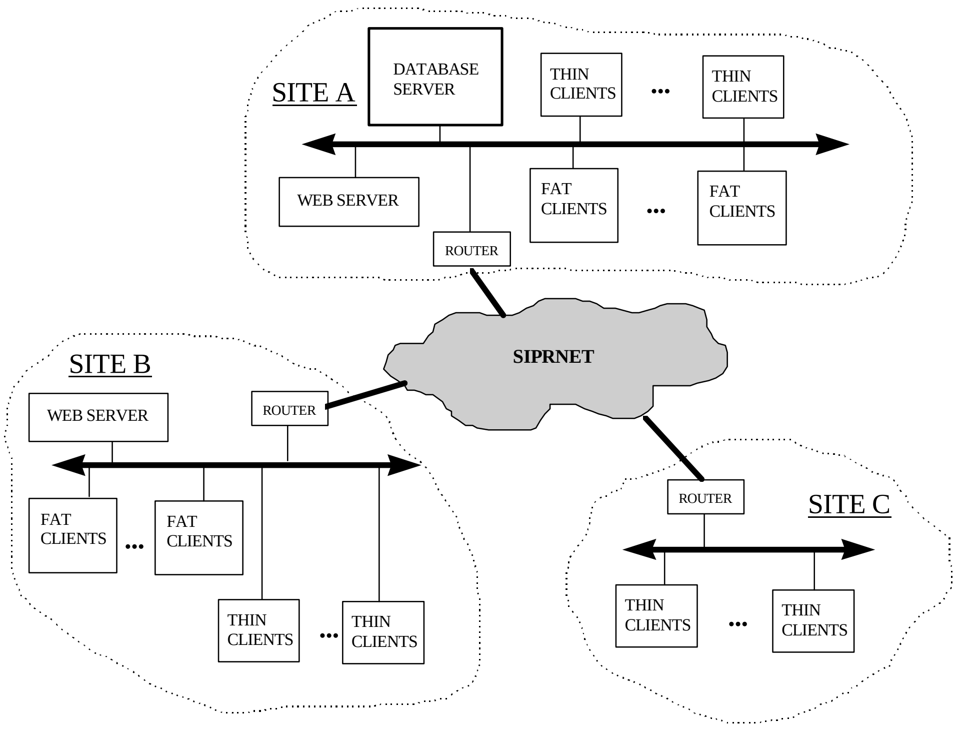


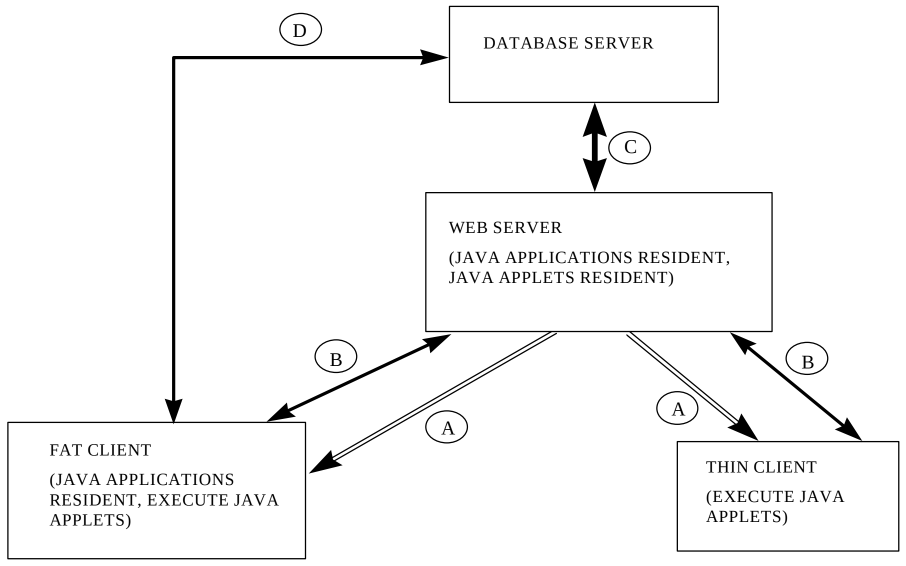
Figure 3-3 shows three types of GCCS sites. Site A is a database site. The role of the database site in the network-centric architecture is the same as the current GCCS implementation. A goal of the network-centric architecture is to significantly reduce the number of database sites GCCS-wide (i.e., in theory, the number of database sites could be reduced to twoCa primary and backup). Reduction in the number of database sites implies sufficient bandwidth on the SIPRNET to handle queries and updates and the implementation of large, capable database machines. The benefit to this approach is less overall system complexity, reduction in the number of skilled database administrators, easier data maintenance (only two sources of data), and simplification in data synchronization (the transaction distribution problem is eased). Site A also implements at least one Web server.

Site B replaces the current GCCS configuration for most sites that currently have a database server machine. The role of the database server becomes that of Web server.

Sites A and B show fat and thin clients. Fat clients will have Java (and/or non-Java) applications resident. A fat client can also download a Java Applet and execute it locally. Thin clients can only download and execute Java Applets.

Site C shows a remote site, that only has PCS. As part of GCCS, this site will have only thin clients. To participate in GCCS, the thin clients must connect to a Web server somewhere in the GCCS net to obtain Applets.

The connectivity of fat clients and thin clients relative to the database server is dictated by the rules of Java. Figure 3-4 illustrates this connectivity.


D
DATABASE SERVER
C
WEB SERVER
(JAVA APPLICATIONS RESIDENT, JAVA APPLETS RESIDENT)
B
B
A
A
FAT CLIENT
(JAVA APPLICATIONS RESIDENT, EXECUTE JAVA APPLETS)
THIN CLIENT
(EXECUTE JAVA APPLETS)

The connectivity to a database for Java Applications and Java Applets on thick and thin clients is as follows:

3.2.1 Sequence for Java Applet on Fat or Thin Client

1) A browser connects to a Web server and transparently downloads Java Applet bytecode.

2) (READ) The Java Applet executes on Java VM, which resides on a thick client. The Applet accesses the database through the Web server.

3) (WRITE) The Java Applet transfers data to a Web server application.

4) Data is transferred between the database and a Web server application. The Web server may use a Java application to connect to the database or a Java-based COTS product.

5) (READ) The Web server application transfers data to a Java Applet, which displays results.

3.2.2 Sequence for Java Application on Fat Client

1) The Java Application connects to database directly like any other application.

3.3 Database Access with Java

Java V1.1 will incorporate a package called *java.sql*, which provides a standard Java binding for accessing an ODBC-compliant database (e.g., Oracle). The *java.sql* package is available now as a beta release from Sun. The major DBMS vendors are also incorporating Java into their products. Some of these are discussed in Section 7.

4.0 THE JAVA LANGUAGE

Java is an object-oriented 3GL. Its originators used constructs from C++; the idea of a virtual machine from SmallTalk; and advanced design aspects from other languages: garbage collection and dynamic linking from Lisp, interfaces from Objective-C, packages from Modula, built-in concurrence from Mesa, and built-in exception handling from Modula-3. The idea behind Java was to build on the power of these languages while trying to step around their shortcomings. Java makes code easier to write and maintain in comparison with older languages.

Java, at first glance, looks like C++ however, there are significant differences between C++ and Java. C++ is an object-oriented extension of C and is fully backward compatible with C. As a result, most programmers do not use the full object-oriented features of C++ and fall back to C. Java, on the other hand, can only be used as an object-oriented language. Java is claimed to be easier to learn and use than C++ and that it will help make object-oriented programming more popular.

The Java language removes some rarely used features of C++ including operator overloading and multiple inheritance. For example, Java does not support the goto statement; instead it provides labeled break and continue statements. The most important simplification is that Java does not use pointers (one of the most bug-prone aspects of C and C++) and implements automatic garbage collection so that the programmer need not worry about dangling pointer references, memory leaks, and memory management.

Java is truly an object-oriented language. This means that the programmer can focus on the data in the application and the interface to it. As compared to C++, Java is more strict in terms of its object-oriented nature. In Java everything must be done via method invocation for a Java object. The whole application must be viewed as an object of a particular class.

Java also supports multi-threading. Multi-threading allows an application to be doing something while waiting for user input. In a GUI-based network application such as Web browsers, there are usually multiple tasks running concurrently. Java provides support for multiple threads of execution that can handle different tasks with a Thread class in the*java.lang* Package. The thread class supports methods to start a thread, run a thread, stop a thread, and check on the status of a thread. This makes programming in Java with threads much easier than programming in the conventional single-threaded C and C++ style.

Java was designed to adapt to an evolving environment and is a more dynamic language than C or C++. Java loads in classes as they are needed, even from across a network. This makes software upgrades much easier and more effective.

4.1 Pre-Defined Java Classes

The default Java environment currently consists of several Java packages that implement a diverse set of fundamental classes which include:

java.lang is the package that contains the basic java classes and native types: Class, Object, Boolean, Float, Double, Integer, String, etc.

java.awt is the Abstract Windowing Toolkit package, which allows the programmer to deal with GUI objects in a generic manner without regard to the system on which the application will run. The classes of this package are roughly divided into three categories: graphics, which define colors, fonts, images, polygons, etc; components, which are classes that define GUI components such as buttons, menus, list, and dialog boxes; and layout managers, which control the layout of components within their container objects.

java.awt.peer is a package consisting entirely of interface definitions. Each *java.awt.peer* interface corresponds to one of the *java.awt* component classes. The interfaces in this package define the methods that must be supported by the GUI components on a specific platform. The Abstract Windowing Toolkit contains methods that create instances of each interface in this package. Normal applications never need to instantiate these peers directly; instead they use *java.awt* component classes, which create peers as needed.

java.applet is the package that contains the Applet class, which is the superclass of all Applets. This class implements an Applet; and to create an Applet, the programmer creates a subclass of this class.

java.io contains classes to support reading and writing streams, files, and pipes. These classes form a fairly structured hierarchy, which are mostly subclasses of the InputStream and OutputStream classes.

java.net contains classes to support network programming. These include dealing with sockets, Internet addresses, network datagrams, uniform resource locators (URLs), and content handlers for data from a URL. The URL class, for example, provides a simple interface to networking. The object referred to by the URL can be downloaded with a single call and the Socket class then allows connection to a specified port on a specified Internet host for read and write data operations using the InputStream and OutputStream classes of the *java.io* package.

java.util contains general purpose utility classes for data structures such as dictionaries, hashtables, dates, stacks, and strings.

4.2 Java Script

Java script is a higher level language scripting language being developed by Netscape (not Sun), which is adapted to use Java bytecode as its output.

5.0 JAVA PERFORMANCE

Performance is clearly an issue in Java-coded applications and especially in Applets. GCCS currently has some large applications that if implemented as Applets, pose several questions such as the time to load across the SIPRNET and the time to execute on the client. The architecture of a software application using Java requires a re-thinking of the fat client approach used by developers in GCCS. Applications will need to be carefully partitioned so that only the GUI components are represented by Applets and transferred across the SIPRNET.

While the issue of sending sophisticated Java Applets though the network will remain an issue that can only be ultimately resolved through careful software design, the issue of Java interpretive execution on the Java VM is being addressed through the use of JIT compilers.

The JIT compilers can be included with the Java VM and can greatly improve the performance by directly turning the bytecodes into machine code. The use of JIT compilers results in Java application execution times comparable to compiled C/C++ coded applications. Java programs will always run slower than native C/C++ coded applications, but with hardware and software improvements the difference may be negligible from a user performance perspective.

6.0 SECURITY

Java has often been labeled as being inherently Anon-secure@ which is an unfair categorization. Java is a computer language, not unlike any other language insofar as security is concerned. Many of the security concerns are directly related to the most-advertised use of Java, namely the transferring of Java Applets over the Internet. More specifically, these concerns are related to the fact that one is downloading someone else=s program to execute on your machine, often without foreknowledge or conscious volition. Clearly, this security concern would be present, no matter what language the downloaded application was written in.

The network-centric nature of Applet-based applications gives rise to other security concerns, as well. The current GCCS security architecture focuses on individual Asites@ as the secure unit. However, it is clear that GCCS is heading toward a fielded architecture in which sites are strongly interdependent for computing resources. There will be many Alightweight@ sites which will be literally incapable of utilizing the GCCS computer system to accomplish their missions unless resources which are located at other sites are available to them. An obvious example of this is the JOPES database, which will only be physically present at (about a dozen) sites, but which must be accessible by all GCCS sites. In this environment, it is crucial that a GCCS-wide security architecture be developed which maintains GCCS system security; promotes inter-site cooperation; and distributes responsibility for system-security to the sites.

The remainder of Section 6 describes how Applet security addressing these two issues will be implemented: Section 6.1 describes mechanisms to ensure that only verified Applets are downloaded for execution, and Section 6.2 describes a distributed security infrastructure to allow secure inter-site computing.

6.1 Executing Authentic DII Applets

When Asurfing@ the web, one often downloads web pages without knowing (or caring) where one is downloading the pages from. This, combined with the transparent nature of Applet invocation, makes Applet-implemented Trojan horses, viruses, imposter programs, etc. a genuine risk. Fortunately, the developers of Java anticipated this danger and designed rigorous security mechanisms to support the dynamic invocation of downloaded Applets. In addition, the widespread appeal of Java has prompted a great deal of security work on the part of the community and as security holes are found, the commercial software developers are responding.

The defense against Abad@ Applets is in two layers. The first layer is a digital signature which can be applied to Applets. The Java Developer Kit (JDK) V1.1 (the Java compiler and debugger package distributed by SunSoft) includes software to generate these digital signatures and attach them to Applets. A web browser with the capability to parse these signatures, such as Netscape Navigator V4.0 (part of the Netscape Communicator), gives the user the ability to specify which digital signatures to honor. Further, the user can specify whether to allow signed applications to execute, or whether simply to allow Applets out of the Asandbox.@

The second layer of defense against bad Applets is the Asandbox@. The sandbox is a set of restrictions placed upon unsigned Applets which prevents them from doing damage to a user=s environment. As is discussed below, these restrictions are so severe that they impede the functionality of a client application. One must keep in mind that the measures described below to establish a sandbox for Applets to execute within and to keep them in the sandbox apply only to unsigned or signed but unauthorized Applets. As described above, Applets bearing an authorized digital signature have the same execution privileges as any application launched on the client machine by the user.

An unauthorized Applet (assuming that it is allowed to execute at all) is not allowed to:

C Read files on the client file system

C Write files on the client file system

C Delete files on the client file system

C Rename files on the client file system

C Create a directory on the file system

C List the contents of a directory

C Check to see whether a file exists

C Obtain information about a file including size, type, and modification of time stamp

C Create a network connection to any computer other than the host from which it originated

C Listen for or accept network connection to any computer connections on any port on the client system

C Create a top-level window without anAuntrusted window@ banner

C Obtain the user=s username or home directory

C Define any system properties

C Run any program on the client system

C Make the Java interpreter exit

C Load dynamic libraries on the client system

C Create or manipulate any thread that is not part of the same ThreadGroup as the Applet

C Create a ClassLoader

C Create a Security Manager

C Specify any network control functions.

In addition to the above security related features, Java relies on three prongs (or layers) of defense: the Bytecode Verifier, the Class Loader, and the Security Manager.

The Bytecode Verifier ensures that the bytecode adheres to the Java rules. A bytecode could possibly be generated by a Ahostile compiler@ that assembled bytecode meant to crash the Java VM. The Bytecode Verifier checks bytecode at a number of different levels. It reconstructs type state information by looking through the bytecode. The types of all parameters of all bytecode instructions must be checked. A key assumption of the Bytecode Verifier is that bytecode may have come from an untrusted source. Because of this possibility the Bytecode Verifier provides a first line of defense against external code that may be trying to break the interpreter. Only code that passes the Bytecode Verifier=s tests will be run.

The Applet Class Loader is the second security defense of Java. The Class Loader determines when and how an Applet can add classes to a running Java environment. Part of its job is to make sure that important parts of the Java run time environment are not replaced by code that an Applet tries to install. The Apple Class Loader, which is typically supplied by the browser vendor, loads all Applets and the classes they reference. When an Applet loads across the network, the Applet Class Loader receives the binary and instantiates a new class. Applets are forbidden to install a new class loader. The Applet Class Loader installs each Applet in a separate name space, which means that each Applet sees its own classes and all of the classes in the standard Java library, but it does not see classes belonging to other Applets.

The Security Manager is the third part of the Java security model and restricts the way an Applet uses visible interfaces. The Security Manager is a single module that performs run time checks on dangerous methods. Code in the Java library consults the Security Manager whenever a potentially dangerous operation is attempted. The Security Manager is defined by the browser vendor through subclassing.

The rules browsers enforce (i.e., Class Loader and Security Manager) make up the security policy defined by a particular vendor. For example, it is possible that Netscape could implement different rules than Microsoft. While many security holes have been discovered in Java Virtual Machines, these have been fixed in later versions of the browsers. Security researchers will continue to find security problems, which in-turn will be fixed by the browser vendors. Although Applets pose a challenge to security, the challenge is not significantly greater than other aspects of network security, and the widespread attention to Applet security can only be viewed as being positive for Java.

6.2 Securing Network-Centric Applications

When Java security was discussed above, it was with respect to avoiding Trojan horse, imposter, and virus Applets (i.e., providing confidence in the validity of the Applets being executed). What was not discussed was how to secure the network-centric system against hostile users. When GCCS upgrades to V3.0, it will have the foundation for a distributed security infrastructure, which it lacks in V2.2. This foundation has four cornerstones: Fortezza, X.500, CORBA, and DCE.

Fortezza is the National Security Agency (NSA) standard for public key/private key security (usually referred to as public key encryption). Public key encryption involves the use of two crypto keys. Any data encrypted with the first key can only be decrypted with the second, and any data encrypted with the second can only be decrypted with the first. Further, the value of one key cannot be deduced from the value of the other, and if these keys are very large (> 10 decimal digits), they cannot be guessed in a reasonable time frame. Each user makes public the value of one of his/her keys (the public key) and keeps the other a secret (the private key). The private key can be used to generate a signature to be attached to some data; other users can then use the public key to identify the author and validate the data. Similarly, an individual=s public key can be used to encrypt data that can then only be read only by the owner of the private key. If the private key is stored on a smart-card (e.g., a Fortezza card), then a user can carry it wherever he goes. If the public key is stored in a network-wide X.500 distributed naming service, then the user carrying the smart card will be able to authenticate his or her identity on any machine on the SIPRNET which supports Fortezza. Further, each user will have an identity that is unique within the entire computer system, and can be authenticated without the user entering passwords, or more importantly, transmitting the password across the network to be authenticated against a password database.

X.500 is the ISO standard for a name service, similar to DNS or NIS. It is intended to be a global name service; a single distributed database accessible from around the world. It is implemented as a hierarchical namespace. Servers are instantiated which take responsibility for a subtree within the hierarchy. As with DNS, queries which cannot be locally resolved are passed on to other X.500 servers until the correct entry is found. Unlike DNS, X.500 is a general-purpose distributed database. The objects storable within the namespace are configurable; attributes can be established for particular classes of objects. Also, the name service is malleable; it is possible for users to modify specific entries in the database in a semi-dynamic fashion. X.500 includes specifications for access control restrictions on the database, providing security for these modifications. X.500 is targeted to two initial DII applications; a white pages for DMS and public key storage for Fortezza. There are many other possible uses for the name service. They key point is that X.500 provides a consistent, secure, universal, cross-platform naming service that can be used to make resources in GCCS available to the users. It can be accessed directly by users via X.500 or LDAP clients (LDAP is the lightweight directory access protocol; it serves as a bridge between TCP/IP based clients and ISO protocol based X.500 servers), or applications can utilize X.500 APIs to access the X.500 directory service.

CORBA and DCE are distributed computing infrastructures. While their philosophies differ, their basic function is the same; they provide the framework upon which distributed programs can be implemented. Both provide an Interface Definition Language (IDL) to define the interfaces between objects (CORBA) or procedures (DCE) running on different computers. In conjunction with the IDL, both CORBA and DCE provide platform-independent communication mechanisms that allow applications to communicate with processes running on another platform without having to be aware of the architecture of the remote platform. The key to this is that both DCE and CORBA provide APIs to a security infrastructure which provides mechanisms to provide authentication, non-repudiation, data validation and encryption. Further, both will soon have the ability to base their security upon Fortezza.

Securing a network-centric system against hostile users, then, would involve the following events. The user inserts his or her Fortezza card into the work station card reader, authenticates against the card and then downloads an Applet and begins its execution. The Applet communicates with a server[1] using either CORBA method invocations or DCE remote procedure calls. The Applet and/or the server with which it is communicating specify via CORBA/DCE APIs the amount and type of security to invoke: authenticate the user (or the application server), verify that transmitted data has not been modified, completely encrypt all communication, etc. CORBA or DCE, in turn, utilizes the Fortezza and X.500 APIs to perform the user authentication, encryption, etc. Thus, Java builds its interprocess communication on top of either CORBA or DCE, which in turn builds its security on top of Fortezza and X.500.

The COTS support for this security architecture either already exists or will exist in the second half of 1997. Iona has a product called *Orbix Web* which is an Aorblet@C a CORBA V2.0 compliant ORB implemented as a Java Applet. This can be downloaded in conjunction with an Applet to provide CORBA communication between the Applet and a server. Also, Netscape is incorporating an ORB into its next version of the

[1] The application server need not be co-located with the web server if the Applet carries a digital signature.

Navigator, thus obviating the need to use Orbix Web (unless non-Netscape web browsers are also in use). Transarc has implemented a Java-to-DCE interface within *Encina* (Transarc=s DCE-based middleware product for client-server application development).

CORBA=s security architecture takes the form of APIs that allow an integrator to implement any security mechanism under CORBA (as long as the mechanism supports a complete set of security functions). The specification for the CORBA V2.0 security architecture was completed in 1996, and COTS ORBs supporting this specification are expected to be available in the latter half of 1997. OSF DCE V1.2 is expected to support Fortezza user authentication.

X.500 compliant name servers are commercially available. X.500 has gained greater acceptance in Europe than in the United States, and thus has a greater installed user base there. However, X.500 is gaining popularity in the States, particularly with LDAP=s increasing use (X.500 provides a strong back end for LDAP). DMS is, in addition to providing X.400-compliant messaging, also providing X.500 service to GCCS.

Fortezza developer=s kits are currently available from the NSA. SAIC currently has several developers who have registered with the NSA and downloaded the development environment. There are multiple vendors selling Fortezza card readers (e.g., Spyrus and Rainbow Technologies). TriTeal has a Fortezza-compliant desktop, and Netscape has announced support for Fortezza in V3.0 of the Enterprise Server (Netscape=s web server). There are also Fortezza-capable X.500 client products available (Chromatix).

This is all relatively new technology. There is certainly a great deal of shaking out yet to be done in terms of which protocols and standards will become accepted by the general marketplace (and which will not); flaws in the current protocol and standard specifications which will themselves have to be cleared up before the COTS software implemented on those specifications are upgraded; and which companies will be able to provide GCCS with the desired products over the long haul.

7.0 RELATED DEVELOPMENTS

The applicability of using Java for mission applications will depend on how much commercial product development occurs to support Java developers. Currently there is a great deal of interest in Java and many claims are being made by the software vendors. It still remains to be determined what is real or will become real versus Avaporware.@

7.1 Interest of the Database Management System (DBMS) Companies

7.1.1 Oracle

According to Oracle, Java is the emerging standard for object-oriented programming in the Internet/intranet arena. Oracle=s strategy is to expand support for mission critical Java applications across all three tiers of the network-centric architecture and to provide essential management and development tools. Oracle=s Java strategy includes delivering a comprehensive set of Java-based servers, tools, and applications across all tiers of the network-centric architecture to support the development of scalable enterprise systems. Oracle has new support for Java J/SQL, which is embedded SQL for Java. J/SQL provides an easy-to-use, high-level interface to relational data. Oracle will also implement Java functionality in the following products in 1997: J/SQL, Thin-Client Java DataBase Connectivity (JDBC), Oracle Version 8, Oracle InterOffice, Oracle Developer/2000, Oracle Designer/2000, Oracle Power Objects, Oracle Discover, Sedona, HatTrick, Oracle Applications, Oracle Discoverer, and Oracle Enterprise Manager.

7.1.2 Informix

Informix is the first company to provide a good solution for enabling customers to build intelligent Web applications and is Netscape=s database of choice for enterprise intranet application development and deployment. Informix=s comprehensive Java strategy is a Java-anywhere solution that allows users to create Web application logic and content in Java and execute it in the client layer, middle layer on an application server, or native database engine. Informix is creating a full complement of Java-aware tools and has been working closely with Sun to ensure that these products are fully Java compatible. Informix products include: J-Works Tool, Web DataBlade Module, and Web Connect for the OnLine Product Family and for the Informix Universal Server.

7.1.3 Sybase

Sybase has been working closely with Sun on defining the JDBC, and its jdbcCONNECT is one of the first full Java implementations of the JDBC. Configurations for jdbcCONNECT are always done centrally on the server without impacting the client and can easily handle large number of concurrent users in a multi-tier environment. Sybase jdbcCONNECT was designed specifically for thin client Java Applets and applications.

7.2 IBM=s San Francisco Project

The IBM San Francisco project is interesting since it is an ambitious effort to build re-useable business objects to support widescale software development and customization. The model and example set by this project could be a future model for GCCS development.

The San Francisco project is to create application frameworks so that the time and effort needed to create new applications will be reduced. The focus of this project is the development of application software for business. The term framework refers to an integrated group of software objects that contain the technical foundation for a particular business application (e.g., General Ledger). A framework is the skeleton of the application and cannot function by itself. It can however, significantly reduce the effort needed to create a complete application.

San Francisco would not be possible without object technology and a very important decision for this project was to use Java. IBM recognized Java as object-oriented and having the best approach to cross- platform development available. IBM strongly supports Java and is investing heavily in efforts to help it evolve. The decision to base San Francisco on Java means that a number of technical issues (which could favorably impact other development such as GCCS) will need to be resolved over time. Even though Java is still evolving, support for IBM's decision to base San Francisco on Java is very strong among developers. Most developers believe that problems will be resolved quickly because there is such a widespread interest in Java.

Software built using the San Francisco application framework will be made up of four layers: Base, Common Business Objects, Core Business Processes, and software created by developers. The lowest layer is called the Base, and it interacts with the operating system of the platform on which it runs through the Java VM. The Base can be thought of as middleware that is needed before object technology can be used above it. The Base includes many functions such as finding objects, tracking their names, controlling access to them, resolving conflicts, security administration, and installation (all of these functions are currently encompassed by DII). The Base layer also includes the Object Model Classes, which are low-level objects from which higher-level objects can be created. They provide a consistent model for building objects while hiding the complexity of the underlying infrastructure for developers. Developers may choose to use only the San Francisco Base and not the upper layers.

The next layer above the Base is called the Common Business Objects. This layer consists of a large number of objects that perform functions commonly needed within a business application (e.g., payment terms, discounts, date and time, currency, address, units of measure, calendar, etc).

The third layer of San Francisco is called Core Business Processes. Each Core Business Process is built for one specific type of application such as General Ledger or Warehouse Management. The Core Business Process layer creates application frameworks that perform complex functions and can also be easily extended.

The final layer is the software created by the developers who buy and extend the San Francisco frameworks. Developers will be able to use other languages as long as linkages to Java are available.

IBM regards San Francisco as a new concept and the extensive use of object-oriented technology makes it unique. The concept of creating frameworks consisting of a software COE (the Base) and common applications started with JMCIS and is currently being extended to DII. The advantage that IBM has in pursuing a similar goal is that it is developing a new system, not replacing legacy systems. The San Francisco project should be looked at by the DII COE developers to see if it has applicability as a future role model for DII.

8.0 RELATED NETWORK-CENTRIC ISSUES

The network-centric GCCS has the following benefits:

- Easy support for multiple platform architectures. Any machine that supports the Fortezza crypto security card, Common Desktop Environment (CDE) and a Java-enabled web browser can be a DII client.

- The ability of any client machine/application to leverage the resources of the entire GCCS in the execution of an task.

- Support for thin clients. As long as a machine has a fast network connection, it requires fewer resources to run, relative to the current fat clients. This is of particular interest for mobile computing, where you perhaps cannot travel with the resources of a full LAN.

- Reduction in the size of the DII COE kernel. Much of the functionality of the DII COE (an operating environment that is common across platforms) is subsumed by the Java Virtual Machine (which, while not yet Aopen@ is certainly a standard) and the CDE (which is open, standard and widely supported). Removing this functionality from the kernel will dramatically reduce the amount of code in the kernel (thus reducing bugs); the degree to which DISA is dependent on any single contractor with respect to the kernel; and the difficulty and time required to install the kernel.

The potential impact of going to thin clients is best illustrated by understanding why the current kernel is not thin.

First, there are components in it that do not, strictly speaking, belong in the kernel. The Common Operational Picture (COP) and Unified Build (UB) are examples. The DII COE will be used throughout the DoD community. For the vast majority of the users, word processing, e-mail, the web, and database access are all that will be required, and if the database clients are implemented in Java, any computer with a word processor, a Fortezza crypto card reader, and a Java-enabled web browser will be satisfactory.

Second, the kernel goes to great lengths to ensure that the platform conforms to the DII COE specifications: applying patches to the operating system, configuring the operating system, setting up the disk drives, etc. By taking responsibility for setting up the base system, DISA acquires responsibilities it cannot keep up with (given the ever increasing numbers of vendor platforms that must be supported), and makes work for itself that it does not have the manpower to accomplish.

Computer vendors, who publish hundreds of kernel patches for each version of their operating system, leave responsibility for patch acquisition and installation to their customers with the recommendation that only those patches necessary for the operation of the user=s particular system be installed. Putting these patches into the kernel both creates a one-size-fits-all kernel configuration and makes DISA responsible for distributing all software patches that may be relevant for any software package installed on a COE system. While DISA should track patches to operating systems and critical software systems, especially patches which impact system security, they should take the same approach as the computer vendors; provide the information, but let the customer be responsible for upgrading their system.

Third, the DII COE kernel still maintains a computer-centric architecture. It loads every resource needed to operate a COE computer onto the computer. Admittedly, if resources such as user login authentication are assumed to be available over the network, and the network goes down, then the computer will be unusable. However, given the network-centric architecture of the GCCS mission applications, it is debatable whether a user could get anything accomplished on a functioning computer that has no network.

If the kernel is reduced to what could be called its essential components; if operating system patches and hardware configuration code are removed from the kernel; and if the COE were re-engineered to take advantage of a network-centric architecture; then the result would be a kernel appropriate for a thin client. This is the first step in creating a strict delineation between client platforms (which require little or no infrastructure beyond the thin kernel), application server platforms (which will require an infrastructure above the thin kernel appropriate to the execution of GCCS mission applications), and database platforms (which will benefit from an infrastructure which optimizes database operation).